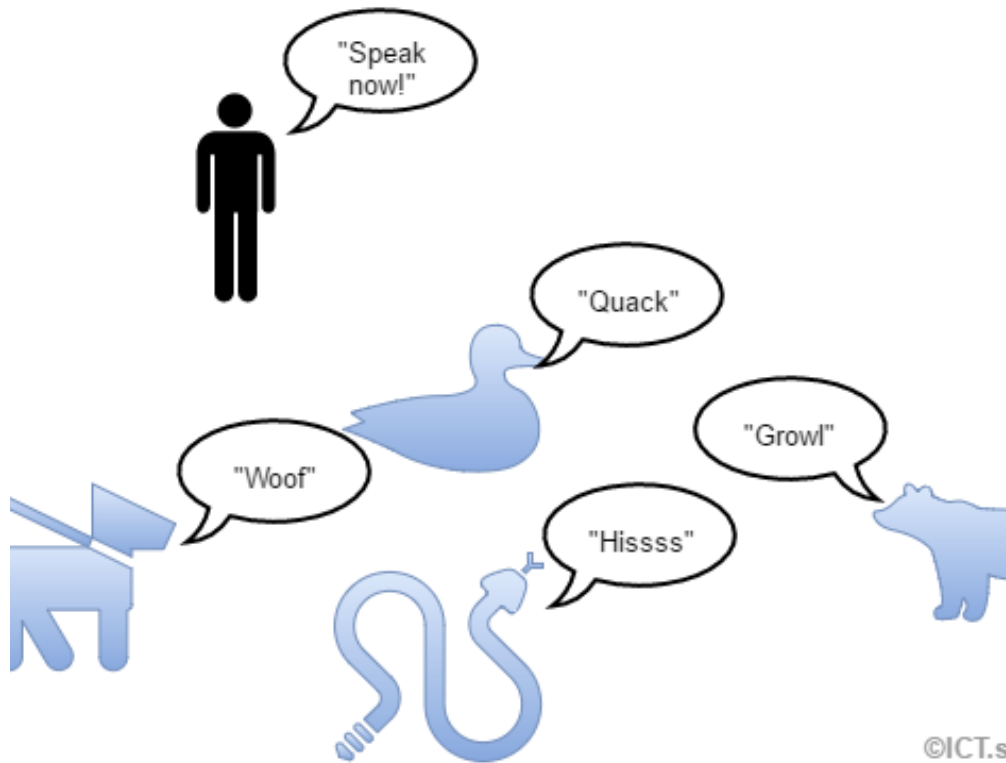


Научно- исследовательск ий практикум

ДИНАМИЧЕСКИЙ И СТАТИЧЕСКИЙ
ПОЛИМОРФИЗМ.

Что такое полиморфизм?



Полиморфизм – средство объектно-ориентированных языков, позволяющее обрабатывать подобным образом объекты разных типов

Например, у нас есть несколько объектов с одинаковым методом `Speak()`

Статический и динамический полиморфизм

ДИНАМИЧЕСКИЙ

- Реализуется при выполнении программы (runtime)
- Способ - за счёт виртуальных функций
- Объявление в базовом классе метода виртуальным

СТАТИЧЕСКИЙ

- Реализуется при компиляции программы (compile time)
- Первый вариант реализации – перегрузка функций (его мы уже знаем)
- В частности, перегрузка операторов
- Второй вариант – использование шаблонов

Давайте вспомним, что же это такое...

Полиморфизм функций:

```
int absValue(int value)
{
    std::cout << "Integer version called!" << std::endl;
    return abs(value);
}

double absValue(double value)
{
    std::cout << "Floating-point version called!" << std::endl;
    return fabs(value);
}
```

Давайте вспомним, что же это такое...

```
int main(int argc, char *argv[])
{
    double absPi = absValue (-M_PI);
    float absE = absValue (-M_E);
    std::cout << absPi << "\t" << absE << std::endl;

    int absTwo = absValue (-2);
    float absTree = absValue (-3);
    std::cout << absTwo << "\t" << absTree << std::endl;
    return 0;
}
```

Давайте вспомним, что же это такое...

```
Starting /Users/amakashov/projects/build-heir-Desktop-Debug/heir...  
Floating-point version called!  
Floating-point version called!  
3.14159    2.71828  
Integer version called!  
Integer version called!  
2          3  
/Users/amakashov/projects/build-heir-Desktop-Debug/heir exited with code 0
```

Или перегрузка операторов

...

Ведь операторы – это
частный случай функций

```
class vector
{
public:
    vector(const double value = 0)
    {
        m_data[0] = m_data[1] = m_data[2] = value;
    }
    vector operator+(const vector& valVector)
    {
        vector ret;
        for (int i=0; i<3; i++)
            ret.m_data[i] = valVector.m_data[i]
+ m_data[i];

        return ret;
    }

    void Print() const
    {
        for (int i=0; i<3; i++)
            cout << m_data[i] << "\t";
        cout << endl;
    }
protected:
    double m_data[3];
};
```

Или перегрузка операторов

...

В результате перегрузки операторов мы можем единообразно складывать как числа, так и вектора – код получается «почти» одинаковым

```
int main(int argc, char *argv[])
{
    double a=0, b=2, c=3;
    vector vec;
    a = c+b;
    cout << "Value of a = " << a << endl;
    vec = vec +c;
    cout << "Vector: ";
    vec.Print();
    return 0;
}
```


Почему такой полиморфизм называют статическим?

- В обоих рассмотренных примерах в момент компиляции программы понятно, какая из функций будет вызвана
- Так, в первом листинге понятно, какой из двух вариантов **absValue** будет вызываться
 - В первых двух строчках – вариант для double
 - Во вторых – для int
- Поэтому полиморфизм связан с этапом компиляции (compile-time)

```
double absPi = absValue (-M_PI);  
float absE = absValue (-M_E);  
std::cout << absPi << "\t" << absE << std::endl;  
  
int absTwo = absValue (-2);  
float absTree = absValue (-3);  
std::cout << absTwo << "\t" << absTree << std::endl;
```

Динамический полиморфизм

- В отличие от статического, какая именно реализация будет использоваться, на этапе компиляции неясно
- Выбрать «правильный» можно только во время выполнения программы (run-time)
- Реализуется с использованием наследования

Оператор switch()

```
switch (value)
{
    case 0:
        ...
    case 1:
        ...
    case 2:
        ...
    default:
        ...
}
```

Здесь value – это целочисленное или перечисляемой (enum) значение

default – если наше значение не совпало ни с одним из реализованным **case**

Но есть неочевидный нюанс


```
switch (1)
{
    case 0:
        cout << '0';
    case 1:
        cout << '1';
    case 2:
        cout << '2';
    default:
        cout << 'd';
}
cout << endl;
```

Результат выполнения будет:

12d

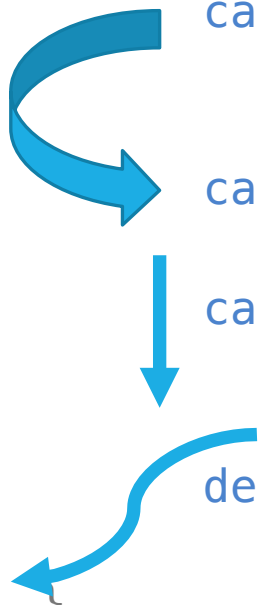
Но есть неочевидный нюанс

```
switch (1)
{
    case 0:
        cout << '0';
    case 1:
        cout << '1';
    case 2:
        cout << '2';
    default:
        cout << 'd';
}
cout << endl;
```



Как это побороть - break

```
switch (1)
{
    case 0:
        cout << '0';
        break;
    case 1:
        cout << '1';
    case 2:
        cout << '2';
        break;
    default:
        cout << 'd';
}
cout << endl;
```



Здесь мы добираемся до

`case 1`

добираемся до

`case 2`

и так как после него стоит

`break`

выходим из

`switch`

Поэтому выведется у нас **12**

Break может использоваться с for...

```
for (int i=0; i<10; i++)  
{  
    cout << i << endl;  
    if (i==5)  
        break;  
}
```

```
Starting /Users/amakashov/pro  
0  
1  
2  
3  
4  
5  
/Users/amakashov/projects/bui
```

...и c while()

```
int i=0;
while(true)
{
    cout << i++ << endl;
    if (i==7)
        break;
}
```

```
Starting /Users/amakashov/projects/
0
1
2
3
4
5
6
/Users/amakashov/projects/build-ope
```


(Лирическое отступление) а ещё есть continue

```
for (int i=0; i<10; i++)  
{  
    if (i<=5)  
        continue;  
    cout << i << endl;  
}
```

```
Starting /Users/amakashov/proj  
6  
7  
8  
9  
/Users/amakashov/projects/buil
```

Пример динамического полиморфизма

Вернёмся к примеру с наследованием, и немного его изменим

Здесь метод **Greetings** определён как виртуальный (virtual)

```
#ifndef PERSON_H
#define PERSON_H

#include <string>
using namespace std;

class Person
{
public:
    Person();
    Person(int age, string name);
    ~Person() {cout << "Person destructor
called for " << m_Name << endl;;}

    int Age() const {return m_Age;}
    string Name() const {return m_Name;}
    virtual void Greetings() const;

protected:
    int m_Age;
    string m_Name;
};
#endif // PERSON_H
```

Реализация

```
#include "person.h"
#include <iostream>

Person::Person()
{
    m_Name = "Anonymous";
    m_Age = -1;
}

Person::Person(int age, string name)
{
    m_Age = age;
    m_Name = name;
}

void Person::Greetings() const
{
    cout << "Hello, guys, my name is "\
    << m_Name << endl;
}
```

Добавим класс наследник

В наследнике метод тоже
является виртуальным

```
class Student : public Person
{
public:
    Student(int age, string name, int mark);
    Student() = default;
    ~Student() {cout << "Student destructor
called for " << m_Name << endl;}

    int Mark() const {return m_AverageMark;}
    virtual void Greetings() const;

protected:
    int m_AverageMark;
};
```

Реализация

```
Student::Student(int age, string name, int mark)
{
    m_Age = age;
    m_Name = name;
    m_AverageMark = mark;
    cout << "Student constructor called for " <<
m_Name << endl;
}

void Student::Greetings() const
{
    cout << "Hello, I'm " << m_Name << " and my
average mark " << Mark() << endl;
}
```

Ещё один класс

И снова метод - виртуальный

```
class Teacher : public Person
{
public:
    Teacher() {cout << "Teacher
constructor called for " << m_Name << endl;}

    Teacher(int age, string name) :
    Person(age, name) {cout << "Teacher
constructor called for " << m_Name << endl;}

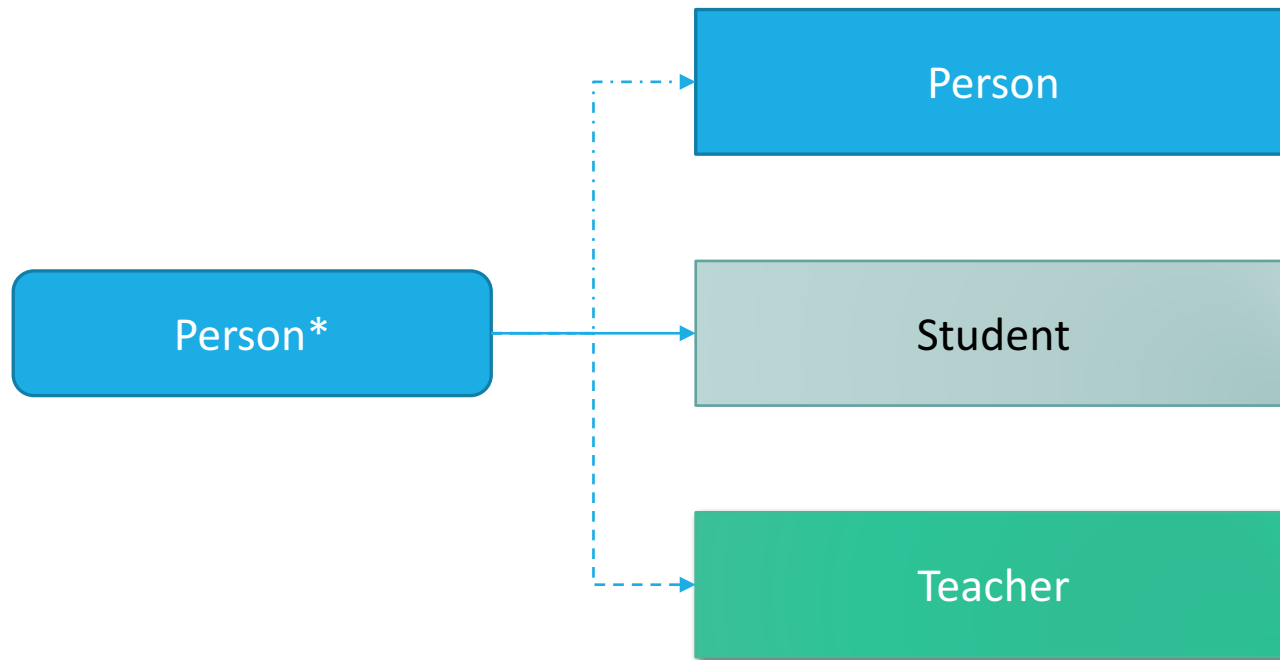
    ~Teacher() {cout << "Teacher
destructor called for " << m_Name << endl;}

    virtual void Greetings() const;
};

void Teacher::Greetings() const
{
    cout << "Hello, students, let's start
out lesson..." << endl;
}
```

Идея с виртуальными функциями

Указатель на объект класса родителя может указывать на функции - наследники



Main.cpp

Здесь создаётся указатель
на класс базового типа,
затем по объекту
вызываются методы

Greetings, Name

Age

```
int main(int argc, char *argv[])
{
    char value = 0;
    Person* ptr = nullptr;
    cout << "Enter 1-3, q to exit" < endl;

    while (value != 'q')
    {
        std::string input;
        std::getline(cin, input);
        value = input[0];
        switch (value)
        {
            ...
        }
        if (ptr)
        {
            ptr->Greetings();
            cout << ptr->Name() << " is " <<
ptr->Age() << " years old" << endl;
            delete ptr;
            ptr = nullptr;
        }
    }
    return 0;
}
```


4TO B switch?

```
switch (value)
{
    case '1' :
        ptr = new Person;
        break;

    case '2':
        ptr = new Teacher (64, "Genadiy");
        break;

    case '3':
        ptr = new Student(22, "Genady-jr.", 4);
        break;

    case 'q':
        break;

    default:
        cout << "Wrong entry " << value << "!
Possible values:\n\t1 - Person";
        cout << "\n\t2 - Teacher";
        cout << "\n\t3 - Student";
        cout << "\n\tq - quit" << endl;
}
```

Результат

Если бы метод был не виртуальным

```
Enter 1-3, q to exit
1
Person constructor called for Anonymous
Hello, guys, my name is Anonymous
Anonymous is -1 years old
Person destructor called for Anonymous
2
Person constructor called for Genadiy
Teacher constructor called for Genadiy
Hello, guys, my name is Genadiy
Genadiy is 64 years old
Person destructor called for Genadiy
3
Person constructor called for Anonymous
Student constructor called for Genady-jr.
Hello, guys, my name is Genady-jr.
Genady-jr. is 22 years old
Person destructor called for Genady-jr.
4
```

Результат

А теперь метод
виртуальный

```
Enter 1-3, q to exit
1
Person constructor called for Anonymous
Hello, guys, my name is Anonymous
Anonymous is -1 years old
Person destructor called for Anonymous
3
Person constructor called for Anonymous
Student constructor called for Genady-jr.
Hello, I'm Genady-jr. and my average mark 4
Genady-jr. is 22 years old
Person destructor called for Genady-jr.
2
Person constructor called for Genadiy
Teacher constructor called for Genadiy
Hello, students, let's start out lesson...
Genadiy is 64 years old
Person destructor called for Genadiy
█
```

Как это достигается?

```
class ExampleClass
{
public:
    ExampleClass() = default;
    int MakeSomething();
protected:
    int a=1, b=2, c=3;
};

int ExampleClass::MakeSomething()
{
    cout <<"a="<< a << " b="<< b << " c="<< c << endl;
}
```

Попробуем посмотреть на ЭТОТ класс:

```
int main(int argc, char *argv[])
{
    ExampleClass example1;
    example1.MakeSomething();
    std::cout << sizeof(example1) << std::endl;
    int* ptr = reinterpret_cast<int*> (&example1);
    for (int i=0; i<(sizeof(example1)/sizeof(int)); i++)
        cout << i <<" : " << ptr[i] << endl;

    return 0;
}
```

Результат

```
Starting /Users/amakashov/projects/build-heir-Desktop-Debug/heir...  
a=1 b=2 c=3  
12  
0 : 1  
1 : 2  
2 : 3  
/Users/amakashov/projects/build-heir-Desktop-Debug/heir exited with code 0
```

Сделаем метод виртуальным

```
class ExampleClass
{
public:
    ExampleClass() = default;
    virtual int MakeSomething();
protected:
    int a=1, b=2, c=3;
};

int ExampleClass::MakeSomething()
{
    cout <<"a="<< a << " b="<< b << " c="<< c << endl;
}
```

Что получилось теперь?

```
Starting /Users/amakashov/projects/build-heir-Desktop-Debug/heir...  
a=1 b=2 c=3  
24  
0 : 89149704  
1 : 1  
2 : 1  
3 : 2  
4 : 3  
5 : 0  
/Users/amakashov/projects/build-heir-Desktop-Debug/heir exited with code 0
```


Построение таблицы виртуальных функций

